HW/ SW Partitioning Algorithms for Multi-objective Optimization in Embedded Systems

Adil Iguider, Oussama Elissati, Abdeslam En-Nouaary, and Mouhcine Chami
Institut National des Postes et Télécommunications, Lab. STRS,
Av. Allal El Fassi, Madinat Al Irfane, Rabat, Morocco.
Email: {iguider, elissati, abdeslam, chami}@inpt.ac.ma

***Abstract*— One of the most crucial steps in the design of modern Embedded Systems (ES) is the partitioning the system's functionalities between the hardware (HW) blocks and the software (SW) blocks. The process of hardware software partitioning (HSP) is driven by several non functional requirements factors. Several works studied the influence of the execution time and the hardware area (cost) factors while dealing with the HSP problem; other works included also the power consumption factor. This article gives a study the HSP problem while considering several factors (ES metrics) and presents two approaches to solve the problem. The first approach has the objective of optimizing simultaneously a number of metrics while respecting a constraint on the global hardware area (cost metric), this approach is implemented using 0-1 Knapsack Problem (KP) algorithm; experimental results show that the algorithm is very fast and gives more reliable solutions comparing to well-known algorithms such as the Genetic Algorithm (GA) and the Simulated Annealing (SA) algorithm. The second approach aims to optimize the hardware cost while respecting given constraints on the other metrics; the proposed approach is based on Balas method.**

***Index Terms*—Embedded Systems, HW/SW Partitioning, Multi-objective, 0-1 Knapsack Problem, Genetic Algorithm, Simulated Annealing, Balas Method.**

I. Introduction

Embedded Systems consist of a combination of hardware components and one or more microprocessors executing software functionalities. The increasing complexity in designing embedded systems involves using high-level system approaches in order to achieve the system functionality and the performance goals. The compound design (Co-Design) is one of the most used approaches to fulfill the latter requirements. The goal is to enable robust system designs and to improve the hardware and the software interactions. The co-design is composed of four parts, co-specification to describe the system functionality at abstract level, co-synthesis to define the system architecture, co-simulation to simultaneously simulate the hardware and the software before prototyping, co-verification to verify mathematically that the system meets the requirements. The Hardware Software partitioning (HSP) process is the most important step in co-synthesis part and in the whole co-design process. The HSP consists of splitting the system's tasks into two parts, a hardware part and a software part, and choosing the best tasks to be included in each part based on some specific metrics and requirements.

Most of previous works dealt with the HSP problem with two metrics, the performance (execution time) and the cost (hardware area). In fact, the hardware leads to faster speed with more expensive cost while the software leads to cheaper cost with lower speed. The objective is to find the best possible balance between the performance and the cost. Two families of algorithms were proposed. On the one hand, the family of exact solutions such as Integer Linear Programming [1], Branch and Bound algorithm [2], and Dynamic Programming [3], are very successful for an HSP problem of a system with a small number of tasks. However, when the number of tasks increases, exact solutions tend to be slow and become inefficient. On the other hand, the family of heuristic algorithms gives an approximation to the exact solution, and it is very useful for an HSP problem of a system with large size. The most used heuristic algorithms are Simulated Annealing (SA) algorithm [4], Genetic Algorithm (GA) [5], Tabu Search algorithm [6] and Greedy Algorithm [7, 8].

Besides dealing with two metrics, other approaches were proposed to study the HSP problem with three metrics, execution time, hardware cost and power consumption metrics.

In this article, we propose two novel approaches with the objective of optimizing several metrics of the system. The metrics which are taken into account are the execution time, the hardware area, the power consumption, the development time (related to IP reuse), the quality (degree of maturity of the IP), the bandwidth and the memory usage. The first approach aims to optimize simultaneously the involved metrics with the respect of a given constraint on the global hardware area, while the second approach has the objective of optimizing the hardware area with the respect of the constraints on the involved metrics. For the first proposed approach, a weight is associated to each metric and is proportional to the important of this latter; the approach is based on the computation of the benefit of the implementation of the task in hardware; the approach is implemented using 0-1 Knapsack Problem algorithm; to validate its effectiveness,

we compare its results with the results obtained using Genetic Algorithm and Simulated Annealing algorithm. For the second approach, the total of each metric is calculated, and then, the Balas method is used to minimize the total hardware cost without exceeding the constraints on the other metrics.

The remainder of this article is organized as follows. Section II reviews the related works. In section III, we present the first proposed approach. In section IV, we give details of the implementations of this approach. The second proposed approach is presented in section V. Section VI shows the results of simulations. Lastly the conclusions and future works are summarized in section VII.

II. Related Work

The co-design as defined in [9] is the design of cooperating hardware components and software components in a single design effort. Choosing a good balance between hardware implementation and software implementation is driving by several factors. Several works studied the influence of the execution time and the hardware cost while dealing with the HSP problem, either by exact solutions or by heuristic approaches.

The family of exact solutions includes algorithms, such as Integer Linear Programming (ILP), Branch and Bound algorithm (B&B), and Dynamic Programming (DP).ILP formulation consists of a set of variables, a set of linear inequalities, and a single linear function of the variables that serves as an objective function, in [1], the authors proposed to solve the HSP problem using ILP.B&B algorithm is based on binary tree, at each level, each variable can take the values 0 or 1, the objective is to find the path from the top to the bottom of the tree which has the optimal cost under the given constraints; an example of its application to the problem is presented in [2].DP is a method, in which large problems are broken down into smaller problems, and through solving the individual smaller problems, the solution to the larger problem is discovered, an example of using DP in the HSP problem is presented in [3].

The family of heuristic algorithms contains, inter alia, Simulated Annealing (SA) algorithm, Genetic Algorithm (GA), Tabu Search (TS) algorithm, Greedy Algorithm (GR), Hill Climbing (HC) Algorithm and Particle Swarm Optimization (PSO). With the analogy of solid annealing, the Simulated Annealing algorithm is based on a heating phase and a controlled cooling phase. An initial temperature is chosen and a first random solution is generated. Then the temperature is reduced and a neighbor solution is generated iteratively until the end of the cooling phase. In [4] the proposed algorithm is based on a combination of Simulated Annealing and Greedy Algorithm. In [10], the author proposed an enhanced algorithm for Simulated Annealing. Genetic algorithm is based on the survival of the fitness principle, which tries to retain more genetic information from generation to generation. The algorithm generates an initial population and defines a fitness function. The evolution of a generation is based on selection, crossover and mutation steps. The algorithm iterates over those steps and evaluates each generation until certain condition is met and it returns the best individual of the latest generation as solution. In [5], the authors proposed and enhanced GA algorithm with an adaptive method for crossover and mutation steps. Other approaches were based on GA for solving the HSP problem, as in [11, 12]. Tabu Search algorithm picks an arbitrary point and evaluates an initial solution, then it computes the next set of solutions within neighborhood of current solution and picks the best solution from the set and iterates until the optima is reached. TS algorithm accepts non-improving solutions in order to escape from local optimum. In [6], the authors proposed two approaches, one is based on 0-1 Knapsack problem and the other is based on Tabu Search algorithm. Another implementation of the TS algorithm for the HSP problem was presented in [13]. Greedy Algorithm is based on making the locally optimal choice at each stage with the hope of finding the global optimum. In [7], the authors proposed three approaches based on Greedy Algorithm targeted for multi-processor system. In [8], the proposed algorithm is based on Greedy Algorithm and Simulated Annealing algorithm. Hill Climbing Algorithm consists of starting with a sub-optimal solution to a problem, and then repeatedly improves the solution until some condition is maximized; unlike the Greedy Algorithm, Hill Climbing Algorithm has the ability to avoid local minima; in [14], the authors proposed an algorithm based on HC for the HSP problem. Particle swarm optimization consists of a swarm of particles, where particle represent a potential solution (better condition). Particle will move through a multidimensional search space to find the best position in that space; an example of using PSO in hardware software partitioning is presented in [15].Other heuristic approaches were proposed to solve the HSP problem. In [16], the proposed approach is based on bat inspired algorithm. In [17] the authors proposed a methodology on Kernighan/Lin algorithm. In [18] the algorithm proposed is derived from the shortest path computing.

The previous cited articles deal with the HSP problem with two metrics. Other approaches were proposed to study the HSP problem with three metrics, execution time, hardware cost and power consumption metrics. In fact, minimizing the power consumption is particularly crucial for mobile devices and for the cost of the cooling system. In [19], the proposed algorithm is a combination of Genetic Algorithm and Branch and Bound Algorithm to simultaneously optimize the three metrics. In [20], the authors propose two heuristic algorithms to optimize the power consumption under execution time and area constraints for multiprocessor systems. In [21], the proposed algorithm is based on 0-1 Knapsack Problem and Tabu Search algorithm to simultaneously optimize the power and the execution time while respecting the hardware area constraint.

Table 1 summarizes and gives taxonomy of the cited algorithms.

TABLE I
ALGORITHMS IN HW/SW PARTITIONING

Ref.	Based algorithm			Optimized metrics		
	Algo.	Exact	heuristic	Exec time	HW area	Power
1	ILP	*		*	*	
2	B&B	*		*	*	
3	DP	*		*	*	
4	SA		*	*	*	
5	GA		*	*	*	
6	TS		*	*	*	
7	GR		*	*	*	
14	HC		*	*	*	
15	PSO		*	*	*	
16	BAT Inspired		*	*	*	
17	Kernighan/ Lin		*	*	*	
18	Shortest path		*	*	*	
19	GA + B&B		*	*	*	*
20	Tree Partitioning		*	*	*	*
21	KP + TS		*	*	*	*

III. FIRST PROPOSED APPROACH

The goal of the first proposed approach is to optimize several metrics under the total hardware area constraint. In this study, we consider the following metrics: the execution time, the power consumption, the development time, the quality, bandwidth and the memory usage.

The system (ES) is composed of Nblocks, denoted as $B = \{B_1, B_2, B_3, \dots, B_n\}$. Each block B_i can be implemented either on hardware or on software.H denotes the set of blocks to be assigned to hardware and$S$denotes the set of blocks to be assigned to software.

To each block$B_i$considered implemented in hardware, weassign a profitP_i and a weight a_i, the weight is supposed tobe the area of the block in hardware, the profit is functionof the different other metrics: execution time, development time, power consumption, quality, bandwidth, memory usage. The profit of each block is calculated using the formula in the equation below:

$$P_i = \lambda_1.\left(\frac{P_{cons}^{is} - P_{cons}^{ih}}{P_{cons}^{is}}\right) + \lambda_2.\left(\frac{T_{exec}^{is} - T_{exec}^{ih}}{T_{exec}^{is}}\right) + \lambda_3.\left(\frac{T_{dev}^{is} - T_{dev}^{ih}}{T_{dev}^{is}}\right) + \lambda_4.\left(\frac{Q^{ih} - Q^{is}}{Q^{is}}\right) + \lambda_5.\left(\frac{B_d^{ih} - B_d^{is}}{B_d^{is}}\right) + \lambda_6.\left(\frac{M_{em}^{is} - M_{em}^{ih}}{M_{em}^{is}}\right) \quad (1)$$

For each block B_i:

- P_{cons}^{ih} and P_{cons}^{is} is respectively the power consumption in hardware and in software.
- T_{exec}^{ih} and T_{exec}^{is} is respectively the time execution in hardware and in software.
- T_{dev}^{ih} and T_{dev}^{is} is respectively the time of development in hardware and in software, which may involve the IP reuse aspect.
- Q^{ih} and Q^{is} is respectively the quality of the block in hardware and in software.
- B_d^{ih} and B_d^{is} is respectively the bandwidth in hardware and in software.
- M_{em}^{ih} and M_{em}^{is} is respectively the shared memory access in hardware and in software.
- The coefficients λ_1, λ_2, λ_3, λ_4, λ_5, and λ_6 are positive integers used to give more or less importance to the associated criteria.

The objective is to find a partitioning (H, S) for Bsuch that:

$$H \cap S = \emptyset$$

and $H \cup S = B$

With the maximization of the whole profit of the blocks that are implemented in hardware, while respecting the global hardware area constraint$A$:

$$maximize: \sum_{i=1}^{n} P_i * x_i$$

$$subject\ to: \sum_{i=1}^{n} a_i * x_i \leq A \quad (2)$$

Where x_i is the block's implementation (HW=1, SW=0).

IV. IMPLEMENTATION

In this section we implement the proposed approach using three existent heuristic algorithms; the first algorithm is the 0-1 Knapsack Problem algorithm, the second algorithm is the Genetic Algorithm and the third algorithm is the Simulated Annealing algorithm.

A. *0-1 Knapsack Problem algorithm*

The 0-1 Knapsack Problem is defined as follows:

1- Given number of items, where each item has a weight and a profit (integer numbers).
2- Given a knapsack with a maximum capacity (integer number).
3- The problem consists of choosing the items for which the total profit is maximized and the total weight doesn't exceed the knapsack's capacity.

The problem described in the previsions section is a 0-1 Knapsack problem. The items are the blocks of the system and the knapsack's capacity is the hardware area constraint.

The weight of each item (block) is the value of its hardware area, and its profit is calculated using (1).

In the first step, the profit of each block is calculated. If the profit is negative, the block is the moved to the software set (S), elsewhere, the block is candidateto a hardware implementation.

M is the number of blocks candidates to hardware implementation. The pseudo-code ofthe 0-1 Knapsack algorithm is presented in Fig. 1.

1) Input: The array that contains the weights (hardware area)of the blocks $W[1 \dots M]$

2) Input: The array that contains the profits of the blocks $V[1 \dots M]$

3) Input: The capacity A

4) Create a matrix $T[0 \dots M; 0 \dots A]$

5) T *is initialized with 0s*

6) T *is then filled as follows:*

For $i = 0$ *to* M, for $j = 0$ *to* A

a) If $j \leq W[i] : T[i,j] = T[i-1,j]$

b) else:

$T[i,j] = \max(T[i-1,j], V[i] + T[i-1, j-W[i]])$

7) Output: return $T[M,A]$

Fig. 1. 0-1 Knapsack algorithm

The principle of the algorithm is to fill a matrix T in recursive way based on the weights and the profits of the items.

The value of the last line and the last column of the constructed matrix ($T[M,A]$) gives the maximum possible profit. To find the selected items that make this maximum, an additional algorithm is necessary. This algorithm is described in Fig: 2.

1) $i = M; k = A$

2) *while* $i \geq 1$ *and* $k \geq 1$

a) if $V[i,k] \neq V[i-1,k]$:

i) mark the i^{th} *block as selected*

ii) $i = i - 1$

iii) $k = k - W[i]$

b) else:

$i = i - 1$

Fig. 2. 0-1 Knapsack algorithm: selected items

At the end, the selected blocks using KP algorithm are moved to the hardware set, while the unselected blocks are moved to the software set. The two constructed sets compose the solution to the initial problem defined in the previous section.

8) Example

The system is composed of four blocks; the profits and the weight of each block are as follows:

TABLE II
KP PARAMETERS

Block	Profit	Weight
1	3	2
2	4	3
3	5	4
4	6	5

The capacity (hardware area constraint) is $A = 5$, and:

$$W = [2, 3, 4, 5]$$
$$V = [3, 4, 5, 6]$$

Using the algorithm of Fig. 1, the constructed matrix T is as follows:

TABLE III
CONSTRUCTED T MATRIX

0	0	0	0	0	0
0	0	3	3	3	3
0	0	3	4	4	7
0	0	3	4	5	7
0	0	3	4	5	7

Using the algorithm of Fig. 2, the selected blocks are 1 and 2, therefore, the partitioning solution is:

$$S = \{3, 4\}$$
$$H = \{1, 2\}$$

B. Genetic Algorithm

The steps of the Genetic Algorithm are summarized in Fig. 3:

1) Initialize the parameters of the GA
2) generation := 1
3) Randomly generate an initial population of candidate solutions
4) Evaluate the population by computing the fitness of each individual
5) while (termination condition is not met) do:
a) Create a new population
b) for each individual:
i) Select a parent based on its fitness
ii) Apply the crossover on the individual with its parents to produce an offspring
iii) Accept the crossover based on the fitness of the offspring
iv) Apply the mutation on the offspring
v) Accept the mutation based on the fitness of the offspring
vi) Add the offspring to the new population
c) generation := generation + 1
d) Replace the old population by the new population
e) Evaluate the new population
6) end loop;
7) return the best individual

Fig. 3. Genetic Algorithm

An initial population (candidate solutions) is randomly generated. Then, each individual of the population is evaluated, its fitness (cost) is the sum of its hardware blocks profits. The termination test determines whether the process will stop or goes to the next generation. The process stops when the cost of the best solution (individual) for each generation no longer changes for a certain number of generations. When the process continues, a new population (generation) will replace the old one.

To generate a new population, three steps are applied on each individual. First step is the selection of the parent, in which the roulette wheel selection method is used; the individual with the highest fitness has more chance to be selected. Second, the process of crossover is applied to the individual with its parent to produce a new individual (offspring). If the offspring has less fitness or its hardware area is greater than the area constraint, then the crossover is rejected for this individual. The last step is the process of mutation, in this latter, the individual is randomly mutated (randomly selected blocks change the implementation between HW and SW) to generate a new offspring. Again, if the offspring has less fitness or its hardware area is greater than the area constraint, then the mutation is rejected for this individual. Then the algorithm iterates from the evaluation step. The process of elitism can also be applied in order to keep some few individuals with highest fitness from generation to generation.

C. *Simulated Annealing algorithm*

Simulated Annealing (SA) algorithm is based on the analogy between the solid annealing and the combinatorial optimization problem. The steps of the algorithm are summarized in Fig. 4:

1) Construct initial configuration:

$$x^{now} = (HW_0, SW_0)$$

2) Initialize Temperature:

$T = T_i$ */* T_i :Initial Temperature */*

3) $for\ i = 1\ to\ T_l$ *do:* */* T_l : temperature length */*

a) Generate randomly a neighboring solution:

$$x^{next} \in N(x^{now})$$

b) Compute change of cost function:

$$\Delta C = C(x^{next}) - C(x^{now})$$

i) $if\ \Delta C \leq 0\ then$:

$$x^{now} = x^{next}$$

ii) else :

a) Generate q:=random(0,1)

b)$if\ q < e^{\frac{-\Delta C}{T}}$ *then:*

$$x^{now} = x^{next}$$

c) Set new temperature:

$$T = \alpha * T$$

/ α : cooling ratio */*

4) if stopping criterion not met then:

Go to Step 3

5) return solution corresponding to the minimum cost function

Fig. 4. Simulated annealing algorithm

First, the algorithm starts by choosing an initial temperature, and randomly creates an initial solution (a HW set and a SW set). Second, in the generation step, a neighbor solution is created by randomly moving a block of the current solution from one set to the other set. Third, the process of acceptation is applied; the acceptance is function of the energy of the current solution, the energy of the neighbor solution and the actual temperature. The energy of each solution is related to the profit defined in (1). Fourth, the temperature is reduced. Lastly, the algorithm iterates from the generation step until certain condition is met (the energy doesn't change).

V. SECOND PROPOSED APPROACH

The objective of the second proposed approach is to optimize the total hardware area under the constraints on the other metrics. The considered metrics: the execution time, the power consumption and the memory usage. The system is represented in the form of a directed acyclic graph (DAG) as shown in the example of Fig. 5.

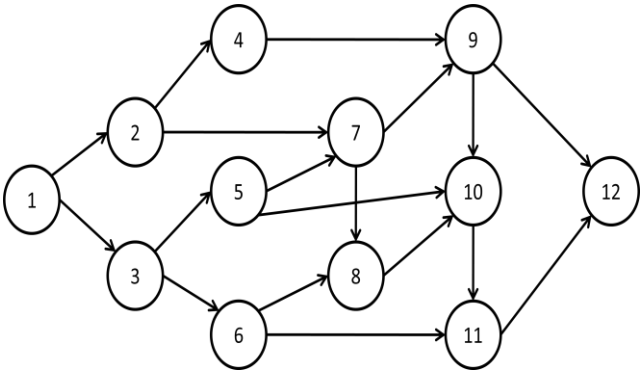

Fig. 5. Example of DAG representation

Each block B_i has the following values:

- P_{cons}^{ih} and P_{cons}^{is}: respectively the power consumption in hardware and in software.
- T_{exec}^{ih} and T_{exec}^{is}: respectively the time execution in hardware and in software.
- C^{ih} and C^{is} : respectively the cost of the hardware area of the block in hardware and in software.
- M_{em}^{ih} and M_{em}^{is}: respectively the shared memory access in hardware and in software.
- x_i represents the block's implementation (in HW = 1, in SW = 0)

As used in [22], the total cost and the total power consumption are given by (3) and (4):

$$C = \sum_{i=1}^{n}[(x_i * C^{ih}) + (1 - x_i) * C^{is}] \qquad (3)$$

$$P = \sum_{i=1}^{n}[(x_i * P_{cons}^{ih}) + (1 - x_i) * P_{cons}^{is}] \qquad (4)$$

Each block B_i is supposed to be executed upon the reception of the corresponding data from its previous block B_{i+1}; therefore, as adopted in [22], the total execution time is the sum of execution time of each block of the system:

$$T = \sum_{i=1}^{n}[(x_i * T_{exec}^{ih}) + (1 - x_i) * T_{exec}^{is}] \quad (5)$$

We consider also that the total shared memory usage of the system is the sum of the shared memory usage of each block of the system:

$$M = \sum_{i=1}^{n}[(x_i * M_{em}^{ih}) + (1 - x_i) * M_{em}^{is}] \quad (6)$$

A. *Problem formulation*

We consider the problem that consists of minimizing the cost of the hardware area with the respect of the constraints on the other metrics:

$$\begin{gathered} minimize: C \\ subject\ to: \\ P \leq P_{max} \\ T \leq T_{max} \\ M \leq M_{max} \end{gathered} \quad (7)$$

Let:

$$C^s = \sum_{i=1}^{n} C_{cons}^{is}$$

$$P^s = \sum_{i=1}^{n} P_{cons}^{is}$$

$$T^s = \sum_{i=1}^{n} T_{exec}^{is}$$

$$M^s = \sum_{i=1}^{n} M^{is}$$

And for each block B_i:

$$c_i = C^{ih} - C^{is}$$

$$p_i = P_{cons}^{ih} - P_{cons}^{is}$$

$$t_i = T_{exec}^{ih} - T_{exec}^{is}$$

$$m_i = M_{em}^{ih} - M_{em}^{is}$$

As C^s is a constant value, the problem of (7) can be formulated as follows:

$$\begin{gathered} minimize: \sum_{i=1}^{n} c_i * x_i \\ subject\ to: \\ \sum_{i=1}^{n} p_i * x_i \leq P_{max} - P^s \\ \sum_{i=1}^{n} t_i * x_i \leq T_{max} - T^s \\ \sum_{i=1}^{n} m_i * x_i \leq M_{max} - M^s \end{gathered} \quad (8)$$

The cost of the hardware area is bigger when the block B_i is implemented in HW; therefore the coefficients c_i are all positives. Consequently, the problem described in (8) can be resolved using the Balas' technique.

B. *Balas Method*

The Balas' algorithm is explained in [23]. Fig. 6 shows a simplified version of the steps of the algorithm.

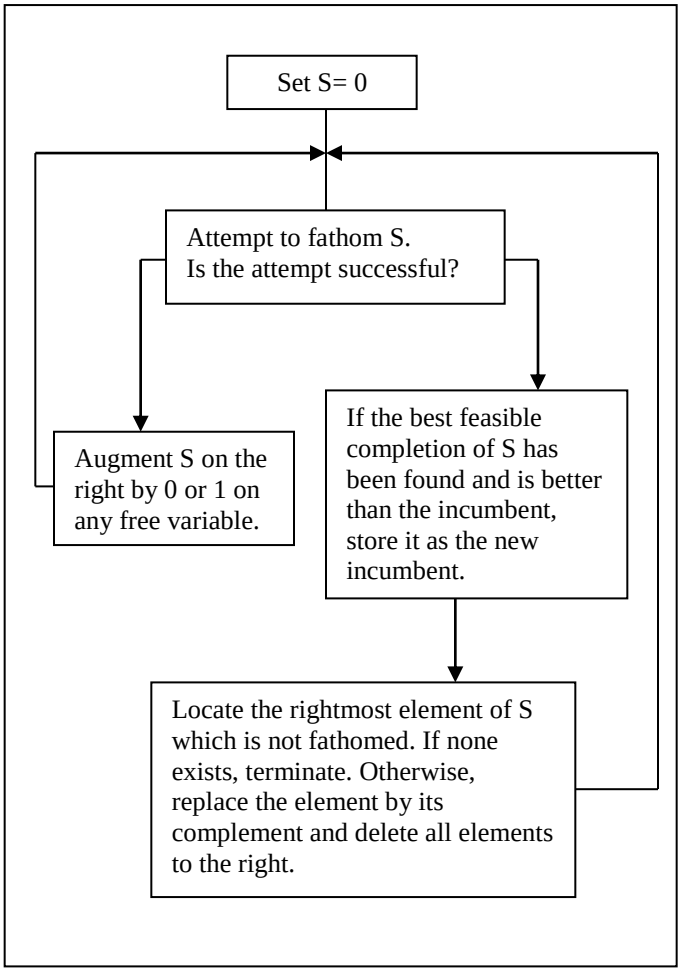

Fig. 6. Simplified version Balas' algorithm

The method requires a LP in canonical form with $c_i \geq 0$:

$$\min z = \sum_{i=1}^{n} c_i * x_i$$
$$x_i \in \{0,1\}$$
$$\sum_{i=1}^{n} a_{ij} * x_i \leq b_j \qquad 1 \leq j \leq m \tag{9}$$

The method is a tree method which progressively fixe to 0 or to 1 the variables x_i.

A vertex S_t of the tree corresponds to a partial solution, for which a subset $F(t)$ of variablesx_i are fixed to 0 or 1.The remaining variables form the set of free variables $L(t)$. In $F(t)$, we distinguish two subsets, the subset of variables fixed to 0 noted $F_0(t)$, and the subset of variables fixed to 1 noted $F_1(t)$. We choose a variable x_i from $L(t)$ and we set it to 0 or 1 which gives two new nodes (children).

For each new created node S_t, the problem (9) becomes as follows:

$$\min z_t = \sum_{i \in L(t)} c_i * x_i + \sum_{i \in F_1(t)} c_i$$
$$\sum_{i \in L(t)} a_{ij} * x_i \leq b_j - \sum_{i \in F_1(t)} a_{ij} = s_j 1 \leq j \leq m \tag{10}$$

1) Node evaluation

After the creation of a new node S_t, it is evaluated. As the objective is to minimize z_t, and as all the coefficients c_i are all positive, the evaluation of the node S_t is as follows:

$$Eval(S_t) = \sum_{i \in F_1(t)} c_i \tag{11}$$

2) Node fathoming

After the creation of a new nodeS_t, the following tests are executed in order:

a. Test 1:
S_t is abandoned if the evaluation of the node is bigger than the provisional solution.

b. Test 2:
S_tIs a terminal node if the evaluation of the node is less than the provisional solution and all s_j are positives.

c. Test 3:
S_t is abandoned if the solution is not realizable. This is the case for which some constraints are not satisfied:

$$\exists j \in [1,m]: \sum_{i \in L(t)} Min(0, a_{ij}) > s_j$$

d. Test 4:
For any constraint (j), if the sum of negative a_{ij} but an a_{pj} is bigger than s_j then x_p is set to 1,and if the sum of negative a_{ij} and a positive a_{pj} is bigger than s_j then x_p is set to 0:

$$if \quad \exists j \in [1,m] \; and \; \exists p \in (t):$$
$$\sum_{i \in L(t)} Min(0, a_{ij}) + |a_{pj}| > s_j$$

Then:

$$x_p = 0 \quad if \; a_{pj} > 0$$
$$x_p = 1 \quad if \; a_{pj} < 0$$

3) The choice of branching

The adopted method is called depth first; the deepest node is developed first. For the separation of branch on a variablex_i, the child node "$x_i = 1$"is treated first. An index i is associated to each fixed variable in the form of "$+j$" if x_i is fixed to 1 and in the form of "$-j$" if x_i is fixed to 0.

Therefore, in the backtrack, the algorithm goes up until it reaches the top level above the variable from which the fathoming was done, if it is negative, it corresponds to a processed child, if it is positive, the algorithm takes the other branch, which corresponds to the complement of the value of that variable, and tries to fathom the new partial solution.

4) Decision of node separation

Let $Q(t) = \{ j \,|s_j < 0 \}$ the set of lines for which the constraint is negative.

And $R(t)$ the set indexes of free variables x_i having at least one negative coefficient a_{ij} from $Q(t)$:

$R(t) = \{i \in L(t), \exists j \in Q(t), a_{ij} < 0\}$.

Let: $P(t) = \sum_{j \in Q(t)} s_j = \sum_{j=1}^{m} \min(0, s_j)$.

The proximity of a solution if x_i is set to 1 is:

$$P(t,i) = \sum_{j=1}^{m} \min(0, s_j - a_{ij}) \tag{12}$$

The choice of the variable of separation i^* is the variable with the maximum proximity value:

$$P(t,i^*) = Max\{P(t,i)/i \in R(t)\} \tag{13}$$

C. Example

In this example, we consider a system (ES) with 5 blocks $B = \{B_1, B_2, B_3, B_4, B_5\}$.

The problem (8) to optimize is as follows:

$$Min: 3x_1 + 5x_2 + 4x_3 + 10x_4 + x_5$$

$$2x_1 - 3x_3 - 4x_4 + x_5 \leq -2 \quad (1)$$

$$-2x_2 - 2x_3 - x_4 \leq -4 \quad (2)$$

$$x_1 - 2x_2 + 2x_3 - x_4 + 3x_5 \leq 4 \quad (3)$$

$$x_i \in \{0,1\}, \quad 1 \leq i \leq 5$$

1) The root node S_0

For the node S_0:

$$L(0) = \{1, 2, 3, 4, 5\}$$

$Q(0) = \{1, 2\}$,

$$R(0) = \{2, 3, 4\}$$

The evaluation of $S_0$is 0. The proximity calculation is given in table IV.

TABLE IV
PROXIMITY CALCULATION FOR S_0

Coefficients and second members						$s_j - a_{ij}$		
x_1	x_2	x_3	x_4	x_5	S_0	$x_2 = 1$	$x_3 = 1$	$x_4 = 1$
2	0	-3	-4	1	-2	-2	1	2
0	-2	-2	-1	0	-4	-2	-2	-3
1	-2	2	-1	3	4	6	2	5
					$P(0)$	-4	-2	-3

The maximum proximity is obtained at x_3, $x_3$is therefore the point of separation. We start by the branch $x_3 = 1$, the branch $x_3 = 0$ will be processed later.

2) The node S_1

For the node S_1:

$$x_3 = 1$$
$$L(1) = \{1,2,4,5\}$$
$$F_1(1) = \{3\}$$
$$Q(1) = \{2\}$$
$$R(1) = \{2,4\}$$

The evaluation of $S_1$is 4. The completion of the other variables ($x_1 = x_2 = x_4 = x_5 = 0$) gives a non-realizable solution as the constraints (2) is violated ($-2 \leq -4$), therefore S_1 is not a provisional solution.

The proximity calculation for $S_1$is given in table V.

TABLE V
PROXIMITY CALCULATION FOR S_1

Coefficients and second members					$s_j - a_{ij}$	
x_1	x_2	x_4	x_5	S_1	$x_2 = 1$	$x_4 = 1$
2	0	-4	1	1	1	5
0	-2	-1	0	-2	0	-1
1	-2	-1	3	2	4	3
				$P(1)$	0	-1

The maximum proximity is obtained at x_2, x_2 is therefore the point of separation. We start by the branch $x_2 = 1$, the branch $x_2 = 0$ will be processed later. The solution $S_1$is then marked with "+j".

3) The node S_2

For the node S_2:

$x_3 = 1, x_2 = 1$

$$L(2) = \{1, 4, 5\}$$
$$F_1(2) = \{2, 3\}$$

The evaluation of $S_2$is 9. The completion of the other variables ($x_1 = x_4 = x_5 = 0$) gives a realizable solution, therefore S_2 is a provisional solution and S_2 is a terminal node.

4) The node S_3

This node corresponds to a backtrack situation, we go back until the top of the branch, in this case we go back until S_0 and we process the branch $x_3 = 0$ and we mark S_0 with "-j". The node S_3 is abandoned as it leads to a non-realizable solution. In fact, the test 3 fails, because for the constraint (2) we have:

$$\sum_{i \in L(t)} Min(0, a_{ij}) > s_j :$$

$$-3 > -4$$

5) The node S_4

We go down until a node marked with "+j". In this case, we process the right hand of S_1 and we mark S_1 with "-j".

For S_4:

$x_3 = 1, x_2 = 0$

$$L(4) = \{1,4,5\}$$
$$F_1(4) = \{3\}$$

$F_0(4) = \{2\}$.

The node S_4 is abandoned as it leads to a non realizable solution. In fact, the test 3 fails, because for the constraint (2) we have:

$$\sum_{i \in L(t)} Min(0, a_{ij}) > s_j :$$

$$-1 > -2$$

6) Termination

The algorithm terminates there is no node to process. The solution found in the node S_2 is the optimal solution:
$x_1 = x_4 = x_5 = 0$ and $x_2 = x_3 = 1$, the evaluation is 9. The developed tree is shown in Fig. 7.

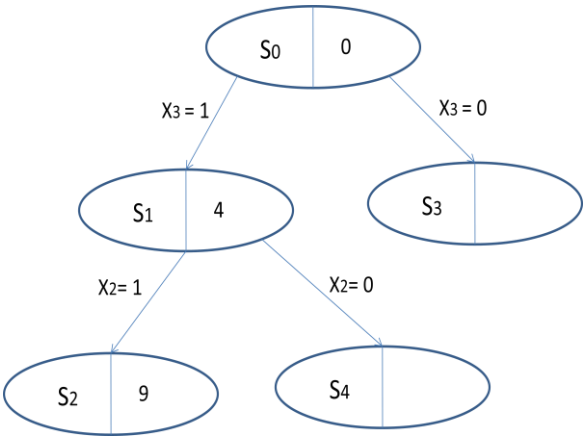

Fig. 7.Tree of Balas' method

The solution of the HSP problem is then:

- The blocks $\{B_2, B_3\}$ are in HW
- The blocks $\{B_1, B_4, B_5\}$ are in SW.

VI. EXPERIMENTS

In this section we give the results of experiments for the first proposed approach.

We implemented the three algorithms (0-1 Knapsack Problem algorithm, Genetic Algorithm, Simulated Annealing algorithm) in Java, and made a number of tests for different number of blocks. The metrics, the coefficients and the hardware area constraint are generated randomly. The metrics of each block are reel numbers and the software execution time of each block is always bigger than the hardware execution time. The coefficients are strictly positive integers. The hardware area constraint is and integer number smaller than the sum of the hardware area of all blocks.

Fig. 8 gives the comparison of the profit obtained using the three algorithm (KP, GA and SA). The results show that the profit obtained using KP algorithm and GA algorithm are almost identical and always bigger than the profit obtained using SA algorithm. GA and SA algorithms fail to converge in some cases, especially with small values of hardware area constraint.

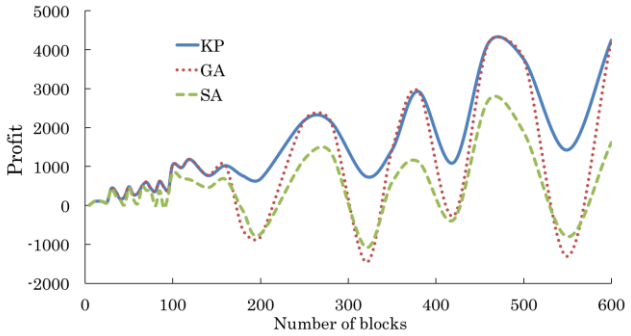

Fig. 8. Comparison between the profits of algorithms

Fig. 9 gives the hardware area obtained for each algorithm. In case of non-convergence, the GA and SA algorithms don't respect the hardware area constraint. This latter is always respected by KP algorithm.

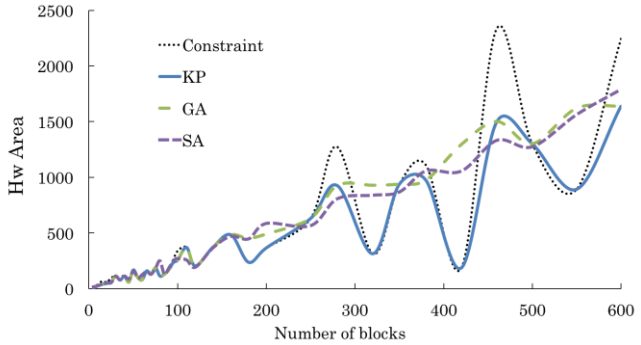

Fig. 9. HW area of algorithms vs. HW area constraint

Fig. 10 gives a comparison between the algorithms in term of their execution time. The results show that KP and SA are very fast comparing to GA algorithm.

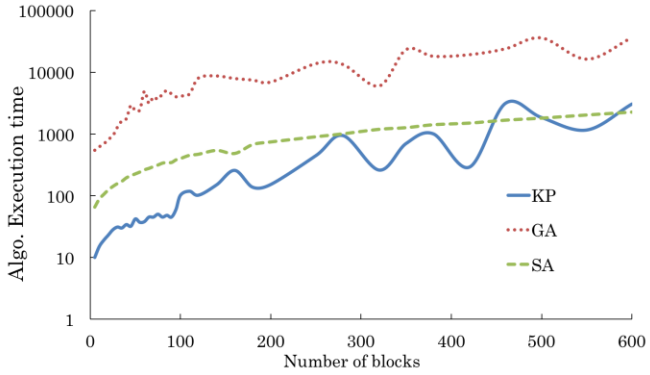

Fig. 10. Algorithms' execution time comparison

VII. CONCLUSION

In this article, we proposed two novel approaches to solve the multi-objective Hardware Software Partitioning problem. The first approach aims to optimize several metrics under a constraint on the global hardware area. The second approach has the objective of minimizing the global hardware area while respecting the constraints on the other metrics. The principle of the first approach is based on the profit obtained when implementing each block in hardware. The profit is a weighted sum function of each metric's benefit when the block is implemented in hardware over its implementation in software. The approach is implemented using 0-1 Knapsack Problem algorithm and compared to Genetic Algorithm and Simulated Annealing algorithm. Experimental results show that the KP algorithm is the best choice to implement the proposed approach; in fact, KP is very fast and gives best solutions comparing to GA and SA algorithms. The principle of the second approach is based on Balas method. As future works, we will implement the second proposed approach, and compare the two approaches with recent algorithms. We will also demonstrate their effectiveness on some practical examples.

REFERENCES

[1] R. Niemann and P. Marwedel, "Hardware/software partitioning using integer programming," in Proceedings of the 1996 European conference on Design and Test, 1996, p. 473.

[2] W. Jigang and S. Thambipillai, "A branch-and-bound algorithm for hardware/software partitioning," in Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology, 2004, pp. 526–529.

[3] P. V. Knudsen and J. Madsen, "Pace: A dynamic programming algorithm for hardware/software partitioning," in Proceedings of the 4th International Workshop on Hardware/Software Co-Design, 1996, p. 85.

[4] Y. Jing, J. Kuang, J. Du, and B. Hu, "Application of improved simulated annealing optimization algorithms in hardware/software partitioning of the reconfigurable system-on-chip," Communications in Computer and Information Science, vol. 405, 2014.

[5] C. W. Jinfu Feng, Junhua Hu and D. Qi, "Hardware/software partitioning algorithm based on genetic algorithm," Journal of Computers, vol. 9, pp. 1309–1315, 2014.

[6] J. Wu, P. Wang, S.-K. Lam, and T. Srikanthan, "Efficient heuristic and tabu search for hardware/software partitioning," The Journal of Super computing, vol. 66, pp. 118–134, 2013.

[7] J. W. Tang, Y. W. Hau, and M. N. Marsono, "Hardware/software partitioning of embedded system-on-chip applications," in proceedings of the IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), 2015, pp. 331–336.

[8] L. Zhang, C. Xu, T. Zheng, and T. Li, "Hardware/software partitioning based on greedy algorithm and simulated annealing algorithm," Journal of Computer Applications, vol. 33, no. 07, p. 1898, 2013.
[9]Chaumont, P. A practical introduction to hardware/software codesign. New York: Springer. 2013.
[10] Banerjee, S. &Dutt, N. Very fast simulated annealing for hw-sw partitioning. Technical Report, CECS-TR-04--17. 2004.
[11]Madhura, P. & Marek R. & Witold P. Genetic algorithms for hardware software partitioning and optimal resource allocation. Journal of Systems Architecture, l (53), 339 – 354. 2007
[12]Knerr, B. &Holzer, M. & Rupp, M. (2007).Novel genome coding of genetic algorithms for the system partitioning problem. International Symposium on Industrial Embedded Systems, SIES'07.134-141.
[13] Lin, G. & Zhu, W. & Ali, M. A tabu search-based memetic algorithm for hardware/software partitioning. Mathematical Problems in Engineering. 2014.
[14]Sim, J. &Mitra, T. & Wong, W. Defining neighborhood relations for fast spatial-temporal partitioning of applications on reconfigurable architectures. International Conference on International Conference on, 121-128. 2008
[15]Farmahini-Farahani, A. & Kamal, M. & Fakhraie, S. & Safari, S. HW/SW partitioning using discrete particle swarm. Proceedings of the 17th ACM Great Lakes symposium on VLSI, 359-364. 2007
[16]Prakasam, S. & Venkatachalam, M. &Saroja, M. &Pradheep, N. &Gowthaman P. A novel bat inspired algorithm for hardware-software codesign partitioning. International Journal of Multidisciplinary Research and Development, l (3), 88-92. 2016
[17]Knerr, B. &Holzer, M. & Rupp, M. HW/SW partitioning using high level metrics. 2004
[18] Wu, J. &Srikanthan, T. & Zou, G. New model and algorithm for hardware/software partitioning. Journal of Computer Science and Technology, l(23), 644-651. 2008
[19] J. Resano, D. Mozos, E. Perez, H. Mecha, and J. Septien, "A hardware/ software partitioning and scheduling approach for embedded systems with low-power and high performance requirements," in International Workshop on Power and Timing Modeling, Optimization and Simulation, 2003, pp. 580–589.
[20] E. Sha, L. Wang, Q. Zhuge, J. Zhang, and J. Liu, "Power efficiency for hardware/software partitioning with time and area constraints on mpsoc," International Journal of Parallel Programming, vol. 43, no. 3, pp. 381–402, 2015.
[21] W. Shi, J. Wu, S.-k. Lam, and T. Srikanthan, "Algorithms for bi-objective multiple-choice hardware/software partitioning," Computers & Electrical Engineering, vol. 50, pp. 127–142, 2016.
[22] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "Hardware/software partitioning with iterative improvement heuristics," in Proceedings of the 9th International Symposium on System Synthesis, 1996, pp. 71–.
[23] A. Geoffrion, "Integer Programming by Implicit Enumeration and Balas' Method", in SIAM Review, Vol. 9, No. 2 pp. 178-190, April 1967